StochDynamicProgramming.jl a Julia library for multistage stochastic optimization

Vincent Leclère¹, Henri Gérard¹, François Pacaud², Tristan Rigaut²

¹ CERMICS, École des Ponts, UPE, Champs-sur-Marne, France
{vincent.leclere,henri.gerard}@enpc.fr
² Efficacity, Champs-sur-Marne, France
{f.pacaud,t.rigaut}@efficacity.com

Mots-clés : dynamic programming, multistage stochastic programming, software library

1 Introduction

StochDynamicProgramming is a Julia Package integrating three algorithms to solve stochastic optimization problems. This package focuses on bringing the Stochastic Dual Dynamic Programming algorithm to non-specialists using a high-level language. Emphasis is put on ease of use, adaptability and documentation. It has minimal dependencies and is distributed under the Mozilla Firefox License encouraging its use in both academic and commercial settings. All information needed to use the package are accessible on GitHub.

The Julia programming language is a high-level, high-performance dynamic programming language for numerical computing becoming more and more popular not only in academic settings but also in industry. It provides a sophisticated just-in time compiler, distributed parallel execution, numerical accuracy, and an extensive mathematical function library. Thanks to its high-level interactive nature and its high-performance results, it is an appealing choice for algorithmic development [1].

StochDynamicProgramming uses this new environment to provide implementation of three well-known methods in Stochastic Programming trying to maintain an easy-to-use interface. This answers the growing need of stochastic optimization by non-specialists (e.g. in the energy industry) to manage systems subject to high variability.

As part of the JuliaOpt organization, the package is available on each platform where Julia is installed, simply using the Pkg.add("StochDynamicProgramming") command in the julia terminal.

2 Underlying technologies

JuMP.jl JuMP is an open-source modeling language that allows users to express a wide range of optimization problems (linear, mixed-integer, quadratic, conic-quadratic, semidefinite, and nonlinear) in a high-level, algebraic syntax. JuMP takes advantage of advanced features of the Julia programming language to offer unique functionality while achieving performance on par with commercial modeling tools for standard tasks [2].

MathProgBase.jl The interface with the different solvers usually used. It allows us to change the solver we want to use changing only one parameter of a function. It also requires a solver package like Cplex.jl or Gurobi.jl.
3 Code capacities

*StochDynamicProgramming* consider controlled dynamic system in discrete finite time (from \([0; T]\)) following a dynamic of the form

\[
x_{t+1} = f_t(x_t, u_t, \xi_t)
\]

(1)

where \(x_t\) is the (perfectly observed) state of the system at time \(t\), \(u_t\) is the control applied to the system at time \(t\), \(\xi_t\) is the exogeneous noise affecting the system between \(t\) and \(t+1\). We assume that \((\xi_0, \ldots, \xi_{T-1})\) is a sequence of independent and finitely supported random variables. We associate at each time step \(t\) a cost function \(l_t(x_t, u_t, \xi_t)\), and a final cost function \(K(x_T)\).

We also defines at each time step \(t\) additional constraints which can be inequality constraints \(g_t(x_t, u_t, \xi_t) \leq 0\) or equality constraints \(h_t(x_t, u_t, \xi_t) = 0\). Finally, we consider the following problem

\[
\min_{x, u} \mathbb{E}_{\mathcal{P}} \left[ \sum_{t=0}^{T-1} l_t(x_t, u_t, \xi_t) + K(x_T) \right]
\]

(2a)

\[
s.t. \quad x_{t+1} = f_t(x_t, u_t, \xi_t) \quad \forall t \in [0; T-1]
\]

(2b)

\[
g_t(x_t, u_t, \xi_t) \leq 0 \quad \forall t \in [0; T]
\]

(2c)

\[
h_t(x_t, u_t, \xi_t) = 0 \quad \forall t \in [0; T]
\]

(2d)

\[
\sigma(u_t) \subset \sigma(\xi_0, \ldots, \xi_t) \quad \forall t \in [0; T-1]
\]

(2e)

The last constraint is the so-called non-anticipativity constraint, specifying the measurability of the control at time \(t\). Note that, for now we only consider risk neutral problem. Extension to risk averse problem are in preparation.

**Extensive Formulation** Considering that the structure of information can be represent on a tree, extensive formulation provides a deterministic problem equivalent to problem (2). Each variable is affected to a node of the tree and the whole problem is given to a solver. This yields an exact solution but the problem becomes quickly untractable.

**Dynamic Programming solver** The dynamic programming solver computes the cost-to-go function recursively backward in time. Unfortunately this requires to discretize states and controls. It is however subject to the curse of dimensionality since problem with state of dimension bigger than 5 are not tractable.

**SDDP solver** The SDDP solver implements a nested and sampled Benders decomposition algorithm [3]. Each iteration contains a forward and a backward loop. During a forward loop, the algorithm simulates a state trajectory of the current approximate policy. Durig a backward loop, the algorithm updates approximation of the Bellman cost-to-go functions along this trajectory. The approximate cost-to-go function in turn define the approximate policy.

The library implement some numerical improvements over the standard SDDP algorithm (cut selection, regularization), as well as some statistical test tools.

**Références**

